TextMetal v4.1.1
project documentation

Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal is released under the MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

Table of Contents

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

The Rationale for TextMetal

Not wanting to leverage heavy, bloated, complex object relational/mapping frameworks (LINQ, Entity Framework, NHibernate, etc.) for several high performance-required applications, the development team originally built TextMetal as a slimmed down fork of the previous Software is Hardwork Library (from the same development team). This allowed the team to focus on the goal of creating a text templating/code generation tool which could generate any textual artifact in general, but with an eye toward domain/data/persistence layer generation in specific. TextMetal is a realization of these goals: the code underlying the templating engine remains mostly generic and constant, while a set of baseline template XML files ship with TextMetal that serve as the "magic spells" which guide the TextMetal tool in generating a fully functional domain/data/persistence layer. Remember however, TextMetal can and does generate any textual artifact for any type of process (software development, build automation, data analysis, etc.).

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Frequently Asked Questions (FAQs)

1. What is a text templating engine (colloquially: "code generator")?
    a. A text templating engine is a program that given a set of inputs produces a correct set of outputs for downstream consumption. Inputs traditionally are text files, XML files, database schema information, etc. Outputs are traditionally XML files, HTML files, program code (C#/VB/C++, etc.), but can be anything useful. These outputs are then consumed by other processes such as compilers, web server scripting interpreters, custom tools, and more.
2. Should I really trust a "code generator" to write production code for my critical application?
    a. Mention "code generation" to most developers and they give you the look like a toddler eating broccoli for the first time. This is only natural; most code generators suck. Some suck less; others suck more. But, at the end of the day, consider this: a software development process that is repeatable and deterministic yields higher quality and less defects. Adding a well-designed code generation process to your overall development strategy can help orient your team to that trajectory. The team must swallow the broccoli; eventually they will learn to like it.
3. If I use a "code generator", does that mean I can save money on developer time?
    a. Possibly. There is an initial investment up front to intelligently and pragmatically integrate code generation into your development process. Factor in template creation, testing, etc. into your project plan or product backlog. Once you catch a stride and assuming your applications are traditional CRUD apps, your team could be generating persistence layers, domain entities, mappings, skeleton UI markup/code, test suites, and more. The sky is the limit; TextMetal is proven to do everything listed here.
4. What runtimes/platforms does TextMetal support?
    a. CLR: Any Windows version that the Microsoft .NET Framework 4.0 supports; Microsoft .NET Framework 3.5 SP1 is supported but requires building from source with a few compiler defines.
    b. Mono: Any Windows, Mac, or Linux version that the Mono supports (including Android for the TextMetal.Core library); requires building from source with a few compiler defines.
5. What other alternatives to TextMetal exist?
    a. CodeSmith Code Generator (commercial)
    b. Microsoft T4 for Visual Studio (commercial)
    c. MyGeneration (open source)
    d. EntitySpaces (commercial)
    e. Code OnTime Generator (commercial)
6. What makes TextMetal unique? Why should I use TextMetal instead of the alternatives?
    a. It is 100% open source, free, with binary and source packages.
    b. Written from the ground up by a world class software architect to solve real world, production "code generation" and build automation problems.
    c. TextMetal supports Mono and Mono for Android.
    d. TextMetal is easily extensible and can be hosted within any application.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

e. TextMetal features a command line host which can be integrated into automated build systems.
f. Source code is legible, fluent, and self-explanatory.
g. Source code is actively maintained and the project has a 3+ year track record in production systems.
h. TextMetal aims to be zero friction: even though it ships with sample (production quality) templates to generate a persistence layer for instance, YOU decide what to generate based on your needs.
i. Fast execution.
j. XML-based, tool friendly template language.

7. Can I integrate TextMetal into my custom application?
    a. Indeed! TextMetal ships with a console host designed for general purpose text/code generation but can be hosted in any application (console, GUI, web, service, etc.) There are samples which provide custom hosting for email templating and even an ASP.NET MVC view engine based host.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Building TextMetal from the Source Code

Visual Studio 2010

1. Ensure that you have Visual Studio 2010 installed.
2. Obtain the source tree per the instructions on the 'Source' tab.
3. Open the solution file '(root)/src/TextMetal.sln' in Visual Studio 2010.
4. Build the solution (Build menu > Build Solution menu item).
5. Ensure that the build is successful in the 'Output' window.

Windows SDK 7.1

1. Ensure that you have the Windows SDK 7.1 installed.
2. Obtain the source tree per the instructions on the 'Source' tab.
3. Double click 'trunk.bat' in the folder '(root)'.
4. This will present a green on black console with .NET tools on the PATH.
5. Type 'b(enter)' to run the build batch script.
6. You may specify the build flavor (debug or release) using the respective flags: -debug | -release
7. Ensure that the build is successful: 'Completed successfully.' should be displayed.

SharpDevelop

1. Ensure that you have the Windows SDK 7.1 installed.
2. Obtain the source tree per the instructions on the 'Source' tab.
3. Either open the solution file '(root)/src/TextMetal.sln' in SharpDevelop.
4. Build the solution.
5. Ensure that the build is successful in the 'Output' window.

MonoDevelop

1. Obtain the source tree per the instructions on the 'Source' tab.
2. Either open the solution file '(root)/src/TextMetal.sln' in MonoDevelop.
3. Build the solution.
4. Ensure that the build is successful in the 'Output' window.

Targeting the .NET Framework v3.5 Profile

1. Edit the '(root)/src/.csproj' files: change the
   `<targetframeworkversion></targetframeworkversion>`
   element values from 'v4.0' to 'v3.5'.
2. Edit the '(root)/src/.csproj' files: append the
   `<defineconstants></defineconstants>`
   element values with 'DEFINE_CLR_VERSION_20'.
3. Edit the '(root)/src/TextMetal.WebHostSample/Web.config' file: Change occurrences of '4.0.0.0' to **either** '2.0.0.0' or '3.5.0.0' depending on context.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Installing from the Package

We do not have pre-built packages as of yet. We plan on creating MSI and NuGet packages in the future. If you would like to contribute to this effort, please send an email.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Running for the First Time

The following prerequisites are required:

1. Make sure you have either built from source or installed using a package.
2. Make sure you have access to a SQL Server 2005+ instance with create database permissions.

Here is how you can see what TextMetal does well:

1. On your SQL Server instance, execute the '(root)\src\TextMetal.WebHostSample\Objects\Model\SQL\setup_sql_server_db.sql' DML script.
2. This will create a new catalog called 'TextMetalWebHostSample' and a few sample schema objects.
3. Open the '(root)\template\AdoNet_Code_Generation' directory.
4. Open the 'adonet_codegen_execute.bat' file in a text editor and change the 'Data Source=(local)' snippet to point to the SQL Server 2005+ instance used previously.
5. The application will crank out stuff. Open the 'output' folder and open the generated solution file 'TMWHS.Model.sln'.
6. Viola! Text templating goodness!

Here is how you can see what TextMetal does well on YOUR "stuff":

1. Delete the 'output' folder created in the previous section.
2. Open the 'adonet_codegen_execute.bat' file in a text editor and change the connection string to point to ANY SQL Server 2005+ instance.
3. Try pointing to a SQL Server 2005+ instance that mirrors a real application in your sphere of influence.
4. The application will crank out stuff. Open the 'output' folder and open the generated solution file 'TMWHS.Model.sln'.
5. Viola! Text templating goodness on YOUR "stuff"!

The above process demonstrated using the SqlServerSchemaSourceStrategy to generate a complete and functional data access layer with unit and integration test suites. This same process is used in numerous mission critical production applications.

NOTE: We support SQLite as well at runtime; schema support is on the roadmap. In the future we expect to provide schema support for SQLite. While we would love to support MS Access (JET), MySQL, Oracle, and PostgreSQL, we do not have infrastructure or bandwidth to commit to this at the moment.

Just remember, TextMetal can generate any kind of text output, leveraging an array of in the box "sources":

- SQL Server 2005+ Schema
- Object (serialized by XmlSerializer)

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

- Object (serialized by XmlPersistEngine)
- Associative Object (serialized by XmlPersistEngine)
- XML Schema (simplified, top level elemental)
- Text File (linear, delimited)
- Public Reflection API Discover
- Any ADO.NET Provider - Data
- Null

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Download and Play with Samples

NOTE: The 'Samples' Git repository has been deleted. We will be rolling samples into the main source tree once other preparation work has been completed.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Generating Persistence Layers (pure ADO.NET)

When provided a SQL Server 2005 or higher database connection, TextMetal will generate a set of C# projects (DomainModel, UnitTests, and IntegrationTests) to serve as the foundation for a generic, high performance data/persistence layer. Within the DomainModel project, pure, clean C# code is generated for: domain classes (tables and views), request/result/response classes for stored procedures (where applicable), domain query classes for the 80/20 scenarios, repository classes, and mapping methods between the object and the database. Mapping uses ADO.NET in a proper and generic manner; deviations from standard behavior (parameter types, SQL dialect, etc.) are delineated on a connection type basis and are selected and used seamlessly and transparently at runtime.

The code that is generated supports SQL Server 2000+, SQL CE 4.0+, and SQLite (Mono.Data.Sqlite AND System.Data.SQLite) providers. All that is needed is to change the target database is to edit the application configuration file: change the connection string (connectionStrings/add/@connectionString) and the (assembly qualified) connection type (connectionStrings/add/@providerName). NOTE: Although MSDN notes that "the providerName is the invariant name of the .NET Framework data provider, which is registered in the machine.config file", TextMetal overloads @providerName to represent the connection type (e.g. System.Data.SqlConnection, et. al). Classes are generated as partial for customization while being re-generatively safe.

Of course, there is trade-offs in this approach: we value clean, debuggable, performant data access layers over the alternative; you do not get LINQ support (yet), out of the box dynamic queries, joins/projections, and if there are custom logic needed, then you have to customize the generated code. The goal was performance over conveniences; the missing features other frameworks offer is known and understood. As this tool matured, adding support for mobile platforms underscored the necessity for clean, debuggable, performant data access code.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Generating Persistence Layers (with LINQ to SQL)

Coming soon!

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

Generating Persistence Layers (with Entity Framework)

We do not have templates for this as of yet. We plan on creating templates in the future. If you would like to contribute to this effort, please send an email.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal Architecture

Coming soon!

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal Out of the Box Constructs

Coming soon!

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal Custom Hosts

Coming soon!

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal Custom Elements

Coming soon!

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)

TextMetal Command Line Reference

TextMetal ships with an out of the box console host. Most users will leverage this tool to generate text artifacts.

```
USAGE: textmetal.exe
                -templatefile:"<filepath>|?"
                -sourcefile:"<filepath>|?"
                -basedir:"<directorypath>|?"
                -sourcestrategy:"<asmqualtypename>"
                -strict:"true|false"
```

Option	Required	Description
templatefile	Yes	A file path to the TextMetal template to use. The path is canonicalized, relative to the current directory.
sourcefile	Yes	A source strategy dependent value used to identify which source to use. If the source strategy expects a file path, then path is canonicalized, relative to the current directory. If the source strategy operates on a database, then it will expect a provider specific connection string instead.
basedir	Yes	The output directory path where output artifacts are rooted. The path is canonicalized, relative to the current directory. If any part of the path does not exist, an attempt to create it will occur. Templates may define sub-directories and files using the OutputScopeConstruct.
sourcestrategy	Yes	An assembly qualified type name indicating the source strategy to leverage. The type must implement the 'TextMetal.Core.SourceModel.ISourceStrategy' interface.
strict	Yes	A boolean value {true | false} indicating whether to enforce strict matching semantics. It is recommended that this be set to 'true'.

TextMetal: Copyright ©2002-2012 Daniel Bullington (dpbullington@gmail.com)